Designing distributed algorithms for mobile computing networks

B. R. BADRINATH, Arup ACHARYA and Tomasz IMIELINSKI
Department of Computer Science, Rutgers University, New Brunswick, NJ 08903, USA

Mobile computing represents a new paradigm that aims to provide continuous network connectivity to users regardless of their location. To realize this aim, it is necessary to design distributed algorithms that explicitly account for host mobility and the physical constraints associated with such networks. This paper presents a operational system model for explicitly incorporating the effects of host mobility with appropriate cost measures. It points out the drawbacks of executing distributed algorithms in this model that are not explicitly designed for mobile hosts. To overcome the resource constraints of mobile hosts, we propose a ***two tier*** *principle for structuring distributed algorithms for mobile hosts so that the computation and communication requirements of an algorithm is borne by the static hosts to the extent possible. In addition, since location of a mobile host could change after initiating a distributed computation and before receiving the result, location management of mobile participants need to be explicitly integrated with algorithm design.*

1 Introduction

A wide spectrum of portable, personalized computing devices ranging from laptop computers to handheld personal digital assistants, has recently been introduced. Their explosive growth has sparked considerable interest in providing continuous network coverage to such *mobile hosts* (MHs) regardless of their location. Mobile hosts have primarily been used as "virtual desktops" enabling remote access to information stored at fixed hosts, e.g. electronic mail and messaging services. However, examples of collaborative applications between mobile users have begun to emerge as well, e.g the Wireless Coyote project [16], and manipulating the state of an electricity network by field engineers equipped with handheld mobile devices[14]. It has also been predicted [29] that with the proliferation of personal, mobile computers, new techniques are required to manage shared data distributed in such computers.

The design of distributed algorithms and protocols has traditionally been based on an underlying network architecture consisting of *static* hosts i.e., the location of a host within the network does not change. Consequently, in the absence of site and link failures, the connectivity amongst hosts in the network remains fixed. Distributed algorithms thus assume a model comprising of a set of processes, executing on static hosts, that communicate by messages over point-to-point logical channels. Each channel may span multiple physical links of the network; this set of links and the hosts at the endpoints of the channel does not change with time. However, this model fails to capture the features and constraints of *mobile* hosts and distributed algorithms designed for this model, therefore need to be restructured to tackle host mobility.

To facilitate continuous network coverage for mobile hosts, a static network is augmented with *mobile support stations* or MSSs that are each capable of directly communicating with MHs within a limited geographical area ("cell"), usually via a low-bandwidth wireless medium. In effect, MSSs serve as access points for a MH to connect to the static network and the cell, from which a MH connects to the static network, represents its current "location". MHs are thereby able to connect to the static segment of the network from different locations at different times. Consequently, the overall network topology changes *dynamically* as MHs move from one cell to another. This implies that distributed algorithms for a mobile computing environment cannot assume that a host maintains a fixed and universally known location in the network at all times; a mobile host must be first located ("searched") before a message can be delivered to it. Further, as hosts change their locations, the physical connectivity of the network changes. Hence, any logical structure, which many distributed algorithms exploit, cannot be statically mapped to a set of physical connections within the network. Second, bandwidth of the wireless link connecting a MH to a MSS is significantly lower than the ("wired") links between static hosts[5, 26]. Third, mobile hosts have tight constraints on power consumption relative to desktop machines [5, 13, 21], since they usually operate on stand-alone sources such as battery cells. Consequently, they often operate in a ''doze mode'' or *volumtarily* disconnect from the network. Lastly, transmission and reception of messages over the wireless link also consumes power at a MH, and

1 A preliminary version of this paper appeared in [6].

so, distributed algorithms need to minimise communication over the wireless links. These aspects are characteristic of mobile computing and need to be considered in the design of distributed algorithms.

This paper focusses on the design of disitributed algorithms in the presence of mobile hosts. It introduces a operational system model of a network with mobile hosts and presents a new set of message costs that is appropriate for this model. It is first shown that if a distributed algorithm, that is unaware of host mobility, is directly executed at the mobile hosts, then it incurs a high "search cost" and violates the constraints on power consumption and usage of the low-bandwidth wireless links. We then propose a *two tier* principle for structuring distributed algorithms to bridge the resource disparity between mobile hosts and the fixed network:

To the extent possible, computation and communication costs of an algorithm is borne by the static portion of the network. The core objective of the algorithm is achieved through a distributed excution amongst the fixed hosts while performing only those operations at the mobile hosts that are necessary for the desired overall functionality.

In conjunction with this principle, we also need strategies to track the location of "migrant" MHs, i.e. MHs that invoke a service from the fixed network at one cell, but move between several locations (cells) before receiving the result. Therefore, to deliver the desired result to a migrant MH, a distributed algorithm must now explicitly incorporate location management of migrant MHs in its design.

The benefits of the two-tier principle and various location-managment strategies for migrant MHs, viz. search, inform and proxy, are illustrated by structuring a token-based logical ring for mobile hosts. It is observed that mobility introduces the possibility of unfair accesses to the token (circulating in the logical ring), and we present a scheme to ensure atmost one access to the token by a MH per traversal of the ring. Lastly, we consider an alternative approach for mutually exclusive access to the token amongst the MHs, that does not require explicit location management. This is achieved by *replicating* token requests at all locations (MSSs).

2 The system model

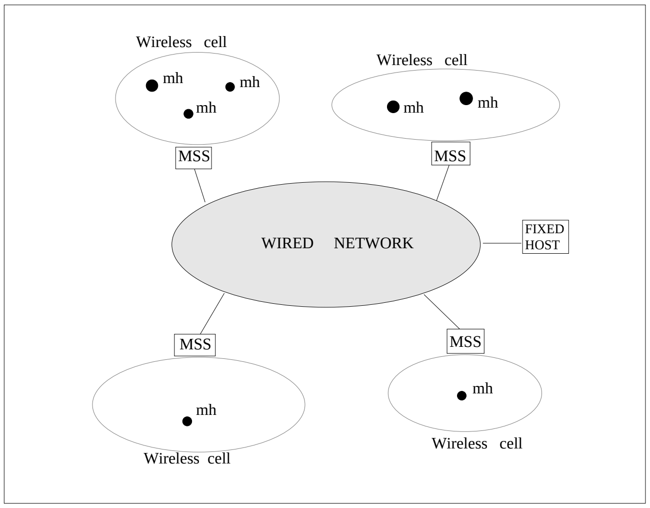

Fig. 1

The term "mobile" implies *able to move while retaining its network connections* [22]. A host that can move while retaining its network connections is a *mobile host* (MH). The infrastructure machines that communicate directly with the mobile hosts are called *mobile support stations* (MSS). A *cell* is a logical or geographical coverage area under a MSS. All MHs that have identified themselves with a particular MSS, are considered to be *local* to the MSS. A MH can *directly* communicate with a MSS (and vice versa) only if the MH is physically located within the cell serviced by the MSS. At any given instant of time, a MH may (logically) belong to only one cell; its current cell defines a MH's "location". In this paper, we assume that all hosts and communication links are reliable. Further, for simplicity of presentation, we assume that all fixed hosts act as MSSs and use the terms MSS and "fixed host" interchangeably.

The system model consists of two distinct sets of entities: a large number of mobile hosts and relatively fewer, but more powerful, fixed hosts (MSSs). The number of MSSs will be denoted by N_{mss} and that of MHs by N_{mh} with $N_{mh} >> N_{mss}$. All fixed hosts and the communication paths between them constitute the *static / fixed* network. A MSS communicates with the MHs within its cell via a wireless medium. The overall network architecture thus consists of a "wired" network of fixed hosts that connect the otherwise isolated, low-bandwidth wireless networks, each comprising of a MSS and the MHs local to its cell. Host mobility is represented in this model as migration of MHs between cells.

The static network provides reliable, sequenced delivery of messages between any two MSSs, with arbitrary message latency. Similarly, the wireless network within a cell ensures fifo delivery of messages between a MSS and a local MH. If a MH did not leave its cell, then every message sent from the local MSS will be received in sequence by the MH. But, since a MH may leave its cell at any time, the sequence of messages received at the MH is a prefix of the sequence of messages sent from the MSS. So, a MH is required to send a *leave(r)* message on the MH-to-MSS channel supplying the sequence number *r* of the last message received on the MSS-to-MH channel. After sending this message, the MH neither sends nor receives any other message within the current cell. Each MSS maintains a list of ids of MHs that are local to its cell; on receipt of *leave()* from a local MH, it is deleted from the list. Conversely, when a MH enters a new cell, it sends a *join(mh-id)* to the new MSS; it is then added to the list of local MHs at the new MSS.

A MH disconnects by sending a *disconnect(r)* message to its local MSS M, where *r* is the sequence number of the last message it received from M (similar to a *leave(r)* message). M deletes the MH from its list of local MHs; however it sets a "disconnected" flag for the particular MH-id. When some other MSS M' attempts to contact this MH (while it is disconnected from the network), M informs M' of the disconnected status of the MH. Later, the MH may reconnect at a MSS N by sending a *reconnect(mh-id, previous mss-id = M)* message. N informs M of the MH's reconnection, and as a result, M unsets the "disconnected" flag for the MH while N adds it to its list of local MHs.

Message communication from a MH h_1 to another MH h_2 occurs as follows. h_1 first sends the message to its local MSS M_1 using the wireless link. M_1 forwards it to M_2, the local MSS of h_2, via the fixed network. M_2 then transmits it to h_2 over its local wireless network. However, since the location of a MH changes with every move and its current location is not universally known in the network, M_1 needs to first determine where h_2 is located before it can forward the message from h_1 to M_2. This is essentially the problem faced by network-layer routing protocols for mobile hosts, such as [9, 22, 28, 30]. Our system model is not tied to any particular routing scheme and instead, we assume that any message destined for a mobile host incurs a fixed *search cost*.

A typical measure of efficiency of a distributed algorithm for fixed networks is the communication complexity of the algorithm, viz. the number of messages exchanged in one execution of the algorithm. However, with the introduction of mobile hosts and their associated resource constraints, the communication complexity must also include the *search cost*, i.e. the number of messages exchanged within the fixed network to *locate* a mobile host. Further, since MHs have tight constraints on power consumption and messages sent on the wireless links require MHs to expend power, communication complexity of a distributed algorithm for MHs should explicitly include the number of wireless messages. Thus, our system model contains three cost measures for counting the number of messages exchanged:

- C_{fixed} – cost of sending a point-to-point message between any two fixed hosts.
- $C_{wireless}$ – cost of sending a message from a MH to its local MSS over the wireless channel (and vice versa).
- C_{search} – cost incurred to locate a MH and forward a message to its current local MSS, from a source MSS. A typical search strategy for a MSS, e.g. [22], would be to query all other MSSs within a search area to determine if a MH is local to its cell. The MSS currently local to the MH will respond, and the original MSS can then forward a data packet to this MSS. Using such a search strategy, where the search cost is linearly proportional to the number of locations (MSSs), C_{search} is equal to $(N_{mss} + 1) \times C_{fixed}$.

Based on the above cost parameters, a message sent from a MH to another MH incurs a cost $2\, C_{wireless} + C_{search}$, while a message sent from a MSS to a non-local MH incurs a cost $C_{search} + C_{wireless}$.

3 Structuring distributed algorithms

Motivation

We cast distributed systems with mobile hosts into a two-tier structure: (1) a network of fixed hosts with more resources in terms of storage, computing and communication, and (2) mobile hosts, which may operate in a disconnected or doze mode, connected by a low-bandwidth wireless connection to this network. The guiding principle for structuring distributed algorithms for MHs in this model is that *the computation and communication demands of a algorithm should be satisfied within the static segment of the system to the extent possible*. Below, we present justifications for this choice:

- ☐ A message sent from a MSS to a non-local mobile host incurs a *search cost*. The same is also true for a message exchanged between two mobile hosts in different cells. To reduce the search component of the overall execution cost, it is desirable that communication between a fixed host and a mobile host occur locally within the same cell.
- ☐ The ability of a MH to operate while on the move, requires a stand-alone source of power viz., batteries. Given the limited life of batteries, power consumption is a serious practical consideration at a MH[5, 13,

21]. In addition to disk accesses and cpu operations, a MH has to expend its limited power resources to send and receive wireless messages; such a constraint is not faced by messages exchanged between fixed hosts over the wired network. Additionally, wireless channels have a significantly lower bandwidth than those within the fixed network. Thus, the number of wireless messages exchanged in any algorithm execution should be minimal, possibly at the expense of a higher number of messages exchanged within the fixed network.

The two points mentioned above suggest that the data structures encapsulating the "state" of an algorithm execution, should mostly reside at the fixed hosts; thus, messages generated to update these data structures will be addressed to fixed hosts. Communication necessary to execute an algorithm may be split into three components: *global*, *local* and *search*. The global component consists of messages whose source and destination are both fixed hosts, e.g. to update appropriate data structures, and mostly represents the communication necessary for the progress of an algorithm execution. The local component refers to communication within a single wireless cell between a MH and its local MSS, and will often be used to initiate an algorithm execution from a MH or to communicate the final result of an execution from a MSS to a local MH. The search component consists of messages that the fixed hosts exchange to determine the current location of a MH so that a message addressed to this MH, may be forwarded to the appropriate MSS. Thus, our approach suggests that the global component dominate the overall communication.

- ☐ The two unique modes of operation of mobile hosts viz., disconnected and "doze-mode", provide compelling arguments against executing an algorithm directly on MHs. When operating in a doze-mode, the MH shuts/slows down most of its system functions to reduce power consumption, and only listens for incoming messages. Like disconnection, this is a voluntary operation. However, the implications are different. In doze mode, a mobile host is *reachable* from the rest of the system and thus, can be induced by the system to resume its normal operating mode, if required. In contrast, disconnection and subsequent reconnection is initiated from the mobile host; it is cut off from the system in the intervening period.

  – A distributed algorithm designed for the mobile computing environment, should not require each MH to participate in every execution of the algorithm. Otherwise, it prevents those MHs from operating in a doze-mode that neither initiated the computation nor is the result of an execution significant to them and consequently, attempts at conserving power by operating in doze-mode are completely thwarted. Thus, by downloading most of the communication and computation requirements to the fixed segment of the network, the static hosts are responsible for the progress of an algorithm execution and a mobile host will not be required to intervene unless it is interested in the outcome of the execution.

  – Algorithms that directly execute at the mobile hosts need to consider the possibility that one or more of the participants may disconnect while an execution of the algorithm is in progress. This has a two-fold effect: (1) the algorithm should be designed to handle a variable number of participants while an execution is in progress, and (2) a search overhead will be incurred if the remaining (mobile) participants need to be informed of a MH's disconnection (and again of its subsequent reconnection). Though distributed algorithms for static systems that are designed to be fault-tolerant, do handle changes in the number of participants, it is inefficient to tackle disconnections of mobile hosts using these algorithms. Disconnection is a voluntary operation and therefore, a MH may inform the system *prior* to an impending disconnection: thus, disconnection should not be associated with the same semantics as failure. The two-tier principle makes it easy to handle disconnections: since the fixed hosts are responsible for the progress of an algorithm execution, disconnection of one or more MHs does not alter the number of participants in the algorithm. Further, prior to disconnecting from the network, a MH can download any data to the fixed network that is necessary for progress of the algorithm.

- ☐ Many distributed algorithms rely on an underlying logical structure such as a ring [24], tree [3] or grid[25], amongst the participants to carry out the needed communication. The main purpose of such a structure is to provide a certain degree of order and predictability to the communication amongst the participants; messages exchanged within such structures follow only selected logical paths. Consider now, the effects of the different operational modes of the mobile hosts on a logical structure comprising of MHs.

  – Disconnection of a mobile node may require that the logical structure be reconfigured, resulting in additional message traffic and possible search overhead.

  – Further, a logical structure predefines the sequence of nodes that a message should traverse starting from a given sender to its destination; thus, the intermediate nodes if operating in doze-mode, are forced to resume normal operation to forward such messages.

Thus, the cost of maintaining a logical structure amongst the mobile hosts may override the benefits of using such an underlying structure for algorithm design. Instead, it may be possible to obtain similar benefits by maintaining the logical structure amongst the fixed hosts without experiencing the disadvantages associated with that of mobile hosts.

Structuring a token-based logical ring for mobile hosts: A case study

A fundamental algorithm in distributed systems consists of circulating a *token* amongst participants in a logical ring. Each participant executes as follows:

- *wait receipt of token from its predecessor in the ring*;
- *enter <critical region>, if desired;*
- *send token to its successor in the ring*.

The algorithm trivially satisfies two important properties: (1) mutual exclusion is trivially guaranteed to the current holder of the token, and (2) it allows fair access to the token by allowing each participant to access the token atmost once in one traversal of the ring.

This simple algorithm has been used for diverse purposes such as termination detection [12], mutual exclusion [24], group membership and message-ordering protocols [15] in fixed systems. In a mobile environment, the token can be used as a mechanism for distributed *access* to a shared resource, for reasons of scalability and for shielding the resource manager from the effects of mobility, e.g. for a database maintained within the fixed network, accessing the token can imply granting a lock to a MH; thus, the logical ring provides a mechanism for distributed lock access to mobile clients without requiring the database's own lock manager to be aware of its clients' mobility.

Algorithm R-MH

Algorithm R-MH is a direct implementation of the logical ring amongst the mobile hosts, and represents the extreme case of executing an existing algorithm without due consideration for host mobility and the associated resource constraints. Although *correctness* of the algorithm is not compromised in this approach, it is insensitive to the resource constraints specific to the mobile computing environment:

- *High search cost* Each message in the algorithm is addressed *to* a mobile host and therefore, incurs a *search cost*. Since the algorithm circulates the *token* amongst all MHs, the overall search cost incurred by the algorithm is proportional to N_{mh}.
- *Excessive usage of wireless links* Both sender *and* destination of *each* message is a MH; the message is thus transmitted over the wireless links between the respective local MSSs of both the sender and destination MHs. Therefore, the wireless cost component of every message in this algorithm is $2 \times C_{wireless}$..

  The cost of each message (to pass the token between two adjacent MHs in the ring) is thus $2 \times C_{wireless} + C_{search}$ and the overall cost of the algorithm is $N_{mh} \times (2 \times C_{wireless} + C_{search})$. Note that this cost is independent of the number of mutual exclusion requests satisfied in one traversal of the ring.
- *Power consumption at MHs* The wireless component of the overall cost is indicative of the power consumed at MHs for transmission and reception of messages. Each message in this algorithm consumes power at both the sender (to transmit it to the local MSS) and the destination (to receive the message from its local MSS). Thus, an execution of the algorithm requires *every* MH to expend power for accessing the wireless link twice, and the cumulative power consumption for one execution of the algorithm is proportional to $2 \times N_{mh}$.
- *Doze and disconnected modes.* Algorithm R-MH requires the participation of *every* MH to maintain the logical ring and cannot therefore permit *any* MH to disconnect. To allow disconnections, the logical ring will need to be reconfigured amongst the remaining participants. Secondly, a MH operating in doze mode, is forced to resume normal operation on receipt of the *token* from its predecessor in the ring. It may revert to doze mode after forwarding the *token* to its successor. Thus, algorithm R-MH does not allow a MH to operate in doze mode *without interruption* even though it does not need to access the shared resource (represented by the token); it must still receive and forward the token to enable other MHs to access the resource.

It is important to emphasize here that the above drawbacks are not intrinsic to the algorithm, but rather stem from an inappropriate application of the algorithm, i.e. the logical ring is established *amongst the mobile hosts*. Unlike fixed hosts, physical connectivity amongst the mobile hosts is redefined on every move and this is manifested through an increase in the search component of the algorithm; further, fixed hosts do not suffer from the constraints of power consumption and low-bandwidth wireless connectivity unlike mobile hosts. We remedy these drawbacks below by applying the algorithm to the mobile environment in accordance with the two-tier principle.

Restructuring the logical ring using the two-tier principle

The two-tier principle suggests that the logical ring should be established within the fixed network. The logical ring now consists of all MSSs with the token visiting each MSS in a predefined sequence. A MH that wishes to access the token is required to submit a request to its local MSS. When the token visits this MSS, all pending requests are serially serviced. However, a MH may have changed its location since submitting its request, and therefore, the location of such migrant MHs need to be explicitly managed. Below, we present two location management strategies, viz. *search* and *inform*, for the case when all MSSs constitute the logical ring.

An alternative method of structuring the ring is to partition the set of all MSSs into "areas" and associate a designated fixed host, called a *proxy*, with each area. The token now circulates only amongst the proxies, and each proxy is responsible for servicing token requests from MSSs within its area. A combination of search and inform strategies is used to manage the location of migrant MHs in this case.

SEARCH Strategy

Actions_executed_by_a_MSS_M

☐ On receipt of a request for the token from a local MH, M adds the request to the rear of its *request queue*.

☐ When M receives the token from its predecessor in the logical ring, it executes the following steps:

1. Pending requests from M's *request queue* are moved to the *grant queue*.
2. *Repeat*
   - Remove the request at the head of the *grant queue*
   - If the MH making the request is currently local to M, then deliver the token to the MH over the wireless link.
   - Else, search and deliver the token to the MH in its current cell.
   - Await return of the token from the MH.

   Until *grant queue* is empty.
3. Forward token to M's successor in the logical ring.

Actions_executed_by_a_MH_h

☐ When *h* requires access to the token, it submits a request to its *current* local MSS.

☐ The MSS where *h* submitted its request will eventually send the token to *h*. After *h* accesses the critical region, it returns the token to the same MSS.

The above algorithm assumes that a MH does not submit a second request if its previous request has not yet been serviced. Secondly, when a MH receives the token, it must return it to the sender MSS after accessing the critical region, i.e. it may not disconnect *permanently* after receiving the token.

Correctness sketch Mutual exclusion is trivially guaranteed by the algorithm since atmost one mh may hold the token at any given time. Next, consider why starvation does not occur, i.e. every request submitted by a mh is eventually granted. It needs to be shown the token can not reside forever at a fixed host and thus, it eventually visits every fixed host in the ring. First, note that the maximum number of requests serviced by a MSS holding the token is bounded: only those requests that were made prior to arrival of the token (since the token's last visit) are serviced. When the token arrives at a MSS, contents of the *request queue* are transferred to the *grant queue*; new requests are then added to the request queue. Only those requests that belong to the *grant queue* are satisfied by the MSS. It follows that the *grant queue* contains atmost N_{mh} requests, one per mh.

Communication cost

- Cost incurred by the token for one traversal of the logical ring : $N_{mss} \times C_{fixed}$
- Cost of submitting a request from a MH to its local MSS: $C_{wireless}$
- Cost of satisfying a token request: $C_{search} + C_{wireless}$
  The above cost is incurred for a *migrant* MH since the MSS (holding the token) first needs to search for this MH and forward the token to the MSS currently local to the MH.
- Cost of returning the token from a MH to the sender MSS: $C_{wireless} + C_{fixed}$
  The C_{fixed} component is incurred when the MH returns the token in a cell other than where it submitted its request.
- Thus, the (worst case) cost of submitting and satisfying a single request is $3\ C_{wireless} + C_{fixed} + C_{search}$.

The total cost of satisfying K requests in one traversal of the ring is thus,

$K \times (3\ C_{wireless} + C_{fixed} + C_{search}) + N_{mss} \times C_{fixed}$

It is reasonable to expect that $K \ll N_{mh}$, i.e. the number of MHs requesting access to the token in a *single* traversal of the ring is much less than the total number of MHs.

The benefits of using the two-tier principle can be quantified by comparing the communication costs of algorithm R-MH and the search strategy:

- *Reduced power consumption.* R-MH consumes power at *every* MH to to receive and forward the token while, after pushing the logical ring to the fixed network, only those MHs that access the token expend power , i.e. the power consumption of R-MH is proportional to *2* N_{mh} while that of the search strategy is proportional to *3K*.

- *Fewer wireless messages.* R-MH sent $2N_{mh}$ wireless messages while only *3K* wireless messages are sent when the MSSs comprise the logical ring.
- *Search cost* The total search overhead of R-MH is proportional to the number of MHs and is independent of the number of mutual exclusion requests satisfied per traversal of the ring. In contrast, the search strategy incurs a search overhead to locate only those MHs that access the token; the total search cost is therefore proportional to *K*, and independent of N_{mh}.

In addition to the above quantitative benfits, R-MH is vulnerable to disconnection of *any* MH and a separate algorithm will need to be executed to reconfigure the ring amongst the remaining MHs. In comparison, with the logical ring within the fixed network, disconnection of a MH that does not need to access the token, has no effect on the algorithm execution. Disconnection of a MH with a pending request can be easily handled since a “disconnected” flag is set for the particular MH at some MSS M within the fixed network: when a MSS M’ (where the MH’s token request is pending) attempts to forward the token to the MH, it is informed by M of the MH’s disconnection and M’ can then cancel the MH’s pending reques..

INFORM Strategy

An alternative to the search strategy to locate a migrant MH is to require the MH to notify the MSS (where it submitted its request) after every change in its location till it receives the token.

Actions_executed_by_a_MSS_M

- ☐ On receipt of a request from a local MH *h*, *M* adds a request <*h, M*> to the rear of its *request queue*.
- ☐ Upon receipt of a *inform(h, M’)* message, the current value of *locn(h)* is replaced with M’ in the entry <*h, locn(h)*> in M’s *request queue*.
- ☐ On receipt of the token, M executes the following steps:

```
1. Entries from the request queue are moved to the grant queue.
2. Repeat
     – Remove the request <h, locn(h)> at the head of the grant queue
     – If locn(h) == M, then deliver the token to h over the local wireless link
     – Else, forward the token to locn(h), i.e. the MSS currently local to h, which will transmit it to h over the local wireless cell.
     – Await return of the token from h.
   Until grant queue is empty
3. Forward token to M’s successor in the logical ring.
```

Actions_executed_by_a_MH_h

- ☐ When *h* needs access to the token,
  - it submits a request to its current local MSS, say M, and
  - stores M in the local variable *req_locn*.
- ☐ When *h* receives the token from the MSS *req_locn*, it accesses the critical region, returns the token to the same MSS and then sets *req_locn* to $\perp$.
- ☐ After every move, *h* now includes *req_locn* with the *join()* message, i.e. it sends *join(h, req_locn)* message to the MSS M’ upon entering the cell under M’.
  - If *req_locn* received with the *join()* message is not $\perp$, then M’ sends a *inform(h, M’)* message to the MSS *req_locn*.

Comparison of search and inform strategies To compare the search nad inform strategies, let a MH *h* submit a request at MSS M and receive the token at M’. Assume that it makes *MOB* number of moves in the intervening period. Then, after each of these moves, a *inform()* message was sent to M, i.e. the *inform cost* is $MOB \times C_{fixed}$. In algorithm R-MSS:search, on the other hand, M would *search* for the current location of *h*and the cost incurred would be C_{search}. Thus, the inform strategy is preferable to search strategy when $MOB \times C_{fixed} < C_{search}$ i.e., if *h* changes cells “less often” after submitting its request, then it is better for *h* to inform M of every change in its location rather than M searching for *h*.

PROXY Strategy

The efficiency of search and inform strategies is determined by the number of moves made by a migrant MH. While a search strategy is useful for migrant MHs that “frequently” change their cells, the inform strategy is better for migrant MHs that change their locations less often. We now present a third strategy that combines advantages of both search and inform strategies, and is tuned for a mobility pattern wherein a migrant MH moves frequently between “adjacent” cells while rarely moving between non-adjacent cells.

The set of all MSSs is partitioned into “areas”, and MSSs within the same area is associated with a common *proxy*. The proxy is a static host, but not necessarily a MSS. The token circulates in a logical ring, which now comprises of only the proxies. On receiving the token, each proxy is responsible for servicing pending requests from its *request queue*. Each request in the queue is an ordered pair <*h, proxy(h)*>, where *h* is the MH submitting the request and *proxy(h)* represents the area, i.e. proxy, where *h* is currently located. A MH makes a "wide area" move, when its local MSS before and after the move, are in different areas, i.e. the proxies associated its new cell is different from the cell prior to the move. Analogously, a "local area" move occurs when its proxy does not change after a move. This assumes that a MH is aware of the identity of the proxy associated with its current cell; this could be implemented by having each MSS include the identity of its associated proxy in the periodic *beacon* message.

Actions executed by a proxy P

☐ On receipt of a token request from a MH *h* (forwarded by a MSS within *P*'s local area), the request <*h, P*> is appended to the rear of the *request queue*.

☐ When *P* receives a *inform(h, P')* message, the current value of *proxy(h)* in the entry <*h, proxy(h)*> is changed to *P'*.

☐ When *P* receives the token from its predecessor in the logical ring, it executes the following steps:

```
1. Entries from the request queue are moved to the grant queue.
2. Repeat
     – Delete the request <h, proxy(h)> from the head of request queue.
     – If proxy(h) == P, i.e. h located within P's area,
       then deliver the token to h after searching for h within the MSSs under P.
     – Else, forward the token to proxy(h) (different from P) which will deliver
       the token to h after a local search for h within its area.
     – Await return of the token from h.
   Until grant queue is empty
3. Forward token to P's successor in the ring.
```

Actions executed by MH h

☐ When *h* requires access to the token, it submits a request to its local MSS and stores the identity of the local proxy in *init_proxy*.

☐ When *h* eventually receives the token from *init_proxy*, it accesses the critical region, returns the token to *init_proxy* and then sets *init_proxy* to ⊥.

☐ After a move, *h* sends a *join(h, init_proxy)* message to the new MSS.

– If *init_proxy* is not ⊥, and *init_proxy* is different from the proxy *P'* serving the new MSS, i.e. *h* has made a wide-area move, then the new MSS sends a *inform(h, P')* message to *init_proxy*.

Communication cost Let the number of proxies constituting the ring be N_{proxy}, and the number of MSSs be N_{mss}; the number of MSSs within each area is thus N_{mss} / N_{proxy}. Let MOB_{wide} be the number of wide-area moves made by a MH in the period between submitting a token request and receiving the token; MOB_{local} represents the total number of local-area moves in the same period, and *MOB* is the sum of local and wide area moves.

Prior to delivering the token to a MH, a proxy needs to locate a MH amongst the MSSs *within its area*. We refer to this as a *local search,* with an associated cost $C_{l\text{-}search}$ and formulate the communication costs as follows:

- Cost of one token circulation in the ring: $N_{proxy} \times C_{fixed}$
- Cost of
  - submitting a token request from a MH to its proxy: $C_{wireless} + C_{fixed}$
  - delivering the token to the MH: $C_{fixed} + C_{l\text{-}search} + C_{wireless}$ (the C_{fixed} term can be dropped if the MH receives the token in the same area where it submitted its request).
  - returning the token from the MH to the proxy: $C_{wireless} + C_{fixed}$

  The above costs add up to: $3\,C_{wireless} + 3\,C_{fixed} + C_{l\text{-}search}$
- Informing a proxy after a wide-area move: C_{fixed}
- The overall cost (worst case) of satisfying a request from a MH, including the inform cost, is then

$$(3\,C_{wireless} + 3\,C_{fixed} + C_{l\text{-}search}) + (MOB_{wide} \times C_{fixed})$$

Comparison with search and inform strategies It is obvious that the cost of circulating the token amongst the proxies is less than circulating it amongst all MSSs (as is the case for search and inform strategies) by a factor N_{proxy} / N_{mss}. The cost of circulating the token within the logical ring is a measure of *distribution* of the overall computational workload amongst static hosts. If all three schemes service the same number of token requests in one traversal of the ring, then this workload is shared by N_{mss} in search and inform schemes, while it is spread amongst N_{proxy} fixed hosts under the proxy method. To compare the efficiency of each algorithm to handle *mobility*, we need to consider the communication cost of satisfying token requests.

The three algorithms incur the following costs to satisfy a MH's request for the token:

$3\ C_{wireless} + C_{fixed} + C_{search}$ . . . (search)
$3\ C_{wireless} + C_{fixed} + (MOB \times C_{fixed})$ . . . (inform)
$3\ C_{wireless} + (3 + MOB_{wide}) \times C_{fixed} + C_{l\text{-}search}$ . . . (proxy)

Based on the above cost measures, it can be seen that the proxy scheme performs better than search when:

$(3 + MOB_{wide}) \times C_{fixed} + C_{l\text{-}search} \ < \ C_{fixed} + C_{search}$
i.e., $MOB_{wide} + 2 \ < \ (C_{search} - C_{l\text{-}search}) / C_{fixed}$ . . . (I)

A typical search strategy, e.g. [22], for a MSS would be to query all other MSSs within a search area to determine if a MH is local to its cell. The MSS currently local to the MH will respond, and the original MSS can then forward a data packet to this MSS. If N_{area} is the number of MSSs within the search area, the search cost is equal to $(N_{area} + 1) \times C_{fixed}$. Using such a search strategy, where the search cost is linearly proportional to the number of locations within the search area, formula (I) above reduces to:

$MOB_{wide} \ < \ N_{mss} - (N_{mss} / N_{proxy}) - 2$

i.e., the proxy scheme performs better than search when the number of wide-area moves made by a migrant MH is two less than the total number of MSSs that is *outside* any given area.

Now compare the proxy and inform schemes. The proxy scheme incurs a lower cost to satisfy a token request when:

$(3 + MOB_{wide}) \times C_{fixed} + C_{l\text{-}search} \ < \ (MOB + 1) \times C_{fixed}$
$\equiv \ C_{l\text{-}search} \ < \ (MOB - MOB_{wide} - 2) \times C_{fixed}$
$\equiv \ C_{l\text{-}search} \ < \ (MOB_{local} - 2) \times C_{fixed}$. . . (II)

Using a search strategy based on [22], the cost of a local search equals $(N_{mss}/N_{proxy} + 1) \times C_{fixed}$, and formula (II) above reduces to:

$(N_{mss} / N_{proxy}) + 2 \ < \ MOB_{local}$

i.e., when the number of local-area moves made by a migrant MH is two more than the number of MSS under each proxy, the proxy scheme outperforms the inform scheme.

Ensuring fair access to the token regardless of mobility

In the algorithms that we have discussed so far, there are two entities whose location varies with time: (a) the token, and (b) MHs. This allows for a situation where a MH accesses the token at its current cell, moves to a cell under a MSS that is the next recipient of the token in the logical ring, and accesses the token again. Thus, a MH by virtue of its greater mobility (compared to a stationary MH) could access the token multiple times in one traversal of the ring by the token, while a stationary MH can access the token atmost once. If "fairness" of access amongst all MHs, independent of mobility, is a desirable goal, then additional synchronization mechanisms need to be built into the algorithm to achieve this goal.

Note also that algorithm R-MH, which maintains the logical ring amongst MHs, allows fair access to the token. Therefore, when we establish the logical ring within the fixed network, we need to also preserve the functionality of algorithm R-MH by providing a scheme that will ensure fair access to the token regardless of a MH's mobility.

As an example, consider two MHs *h1* and *h2* initially located in the same cell under MSS M1. Let MSS M2 be the successor of M1 in the logical ring. If both MHs contend for the token at M1, then both requests will be satisfied when the token reaches M1. Now, assume *h2* moves to the cell under M2 and contends for the token. If its request is received at M2 prior to the token reaching M2 from M1, then *h2* can access the token for a second time. In contrast, if *h1* continues to remain in M1's cell, it can again access the token only after the token revisits M1 after completing one traversal of the ring.

The scenario outlined above does not cause starvation, but it increases the delay for a "stationary' MH to access the token. In the worst case, a token request from a MH such as *h1* may be satisfied only after every other MH has accessed the token once from each of the N_{mss} cells, i.e. after a total of $(N_{mh} - 1) \times N_{mss}$ requests have been satisfied.

Below, we present a scheme that is applicable to all three location management strategies, which prevents a MH from accessing the token more than once in one traversal of the ring, i.e. how often a MH can contend for the token is not affected by its mobility (or lack of it).

1. The token is associated with a loop counter (*token_val*) which is incremented every time it completes one traversal.
2. Each MH maintains a local counter *access_count* whose current value is sent along with the MH's request for the token to the local MSS.
3. A pending request is moved from the *request queue* to the *grant queue* at the MSS (or proxy) holding the token, only if the request's *access_count* is less than the token's current *token_val*.
4. When a MH receives the token, it assigns the current value of *token_val* to its copy of *access_count*.

Note that in step (4) above, a MH's *access_count* is reset to the token's current *token_val*; thus, even if a MH has a low *access_count*, it can the access the token only once in a given traversal. With this scheme, the number of token accesses K satisfied in one traversal of the ring is limited to N_{mh} (when the ring comprises of all MSSs), while K could be $N_{mss} \times N_{mh}$ in the absence of this scheme. In effect, this scheme represents a trade-off between "fairness" of token access amongst the MHs and satisfying as many token requests as possible per traversal.

Other alternative criteria for "fairness" are also possible. For example, a MH may be allowed to access the token multiple times in one traversal of the ring, subject to the limitation that its total number of accesses to the token not exceed the current *token_val*. This criterion can be implemented by modifying step (4) above, as follows: the *access_count* of a MH is incremented on every access, instead of being assigned the current value of *token_val*.

4 Substituting location management with data replication

In the algorithms that we developed so far, explicit location management strategies were incorporated into the base algorithm, viz. a token circulating within a logical ring. This was needed to handle migrant MHs. We now consider a totally different algorithm for distributed mutual exclusion that is based on *replicating* requests at all MSSs.

The advantage of using a replication-based approach is that location of migrant MHs do not need to be explicitly managed, since regardless of its current location, the local MSS has a copy of a migrant MH's pending request. However, this also introduces a problem that if a MSS is presently servicing a MH's request, no other MSS should attempt to service another request; otherwise, mutual exclusion will be violated. This requires that requests from MHs be *globally* ordered amongst all MSSs, and only the first request in this ordered sequence be serviced at any time.

The problem of creating a total order of delivery of messages also arises in static systems, viz. all common recipients of two messages *m* and *n* should either receive *m* before *n* or vice-versa. However, we intend to use the ordering of messages for a different purpose, viz. when all requests are globally ordered, then only the request at the head of this order should be serviced by exactly one MSS. Choice of this MSS is determined by the current location of the MH that made this request: its local MSS can unilaterally decide to service the request with the assurance that no other MSS will simultaneously process any another request.

Below, we present our algorithm for distributed mutual exclusion for mobile hosts that is derived from a message-ordering protocol [10] for static systems.

Actions taken by a MH

- ☐ Each MH *h* maintains a local counter, *h_count*, which is incremented prior to submitting a new request for mutual exclusion. A request for mutual exclusion *req(h, h_count)* is submitted to its local MSS.
- ☐ On a move, *h* includes its current value of *h_count* with the *join()* message to its local MSS.
- ☐ When a MH receives a *grant* message from its local MSS, it accesses the shared resource and then replies with a *release(h, h_count)* message to its current local MSS.

Actions taken by a MSS

- ☐ Each MSS maintains a *delivery queue* of pending requests; each request is flagged as either*deliverable* or *undeliverable*, and is assigned a priority number. The queue is kept sorted in increasing order of message priority numbers.
- ☐ When a MSS M receives *req(h, h_count)* from a MH *h*, it executes a two-phase protocol to order this request relative to all other pending requests.
  - — In the first phase, M sends a copy of *req(h, h_count)* to all other MSSs.
  - — Each MSS assigns a (temporary) priority number to the received request from M, that is greater than any of the requests currently in its queue; the request is tagged *undeliverable* and inserted at the end of the queue. The temporary priority number is sent back to the initiator.
  - — In the second phase, M computes the maximum of all temporary priority numbers, and sends it to all MSSs. Each MSS then assigns this number as the final priority number of the request and tags it *deliverable* and re-sorts the delivery queue.
- ☐ A MSS can service a pending request *req(h, count)* if :
  - – the request is tagged *deliverable,*
  - – the MH *h* is local to its cell, and
  - – the *h_count* value submitted by *h* (either with the *join()* message after entry to its cell, or with the *req()* message to this MSS) is equal to *count*, i.e. the request has not already been serviced by another MSS.

If these conditions are satisfied, then the MSS sends a *grant()* message to *h*.

- ☐ On receiving a *release(h, h_count)* message from a local MH, a MSS first deletes the entry *req(h, h_count)* from its *delivery queue*. It then sends a *delete(h, h_count)* message to all other MSSs, which then delete the corresponding entry from their respective *delivery queues*.

Correctness sketch Correctness of the message-ordering portion of the above algorithm follows directly from the correctness of the two-phase ABCAST protocol of [10], and is not discussed. In the ABCAST protocol, a message is removed from the *delivery queue* of any participant when it reaches the head of the queue and is tagged *deliverable*. However, in our case, a message is a request for mutual exclusion on behalf of a MH, and is "delivered" to the MSS that is currently local to this MH. Hence, we need an explicit *delete* message from this MSS to all other MSSs. Also, we assign a counter value with each request to avoid a MH receiving *grant* messages from more than one MSS for the same request, as shown below:

– A MH *h*, currently local to MSS M_1, receives the *grant* message from M_1 (since the MH's request is at the head of M_1's queue) and replies with the *release* message to M_1. M_1 sends a *delete* message to all other MSSs.

– Before the *delete* message reaches a MSS M_2, *h* moves to M_2's cell. The request from *h* is still in M_2's queue; if the request is at the head of the queue, then M_2 will send a *grant* message to *h*. Thus *h* may receive more than one *grant* message for the same request.

Communication costs It is easy to see that each execution of the above algorithm incurs a total cost of $(4N_{mss} - 1) \times C_{fixed} + (3\ C_{wireless})$. The interesting aspect of this cost is that it involves no search component, i.e. the cost is independent of a MH's mobility. The reason is that a request from a MH is replicated at all possible locations, and therefore, a MSS is always available to service a MH's request locally. However, even though the algorithm requires no explicit search, a higher C_{fixed} cost is incurred to replicate and order token requests.

5 Related Work

The concept of designing distributed algorithms that explicitly cope with host mobility, is still in its infancy. An overview of the impact of host mobility on distributed computations is presented in [7], while [6] contains preliminary ideas that have been fully developed in this paper.

The implications of mobile for distributed data management are considered in [20, 19, 4, 8]. Other related work includes addressing schemes and protocols at the network-layer for routing messages to and from mobile hosts [9, 22, 28, 30], while [11] quantifies the effects of host mobility on transport-layer connections. Application–layer protocols to deliver multicast messages to mobile recipients from exactly-one location are presented in [2, 1]. The design of distributed filesystems has also been influenced by mobility of users, as exemplified by [17, 18, 23, 27]. Various operating systems issues related to mobile hosts can be found in [26, 5, 13, 31].

6 Conclusions

The design of algorithms for distributed systems and their communication costs have been based on the assumptions that the location of hosts in the network do not change and the connectivity amongst the hosts is static in the absence of failures. However, with the emergence of mobile computing, these assumptions are no longer valid. Additionally, mobile hosts have tight constraints on power consumption and bandwidth of the wireless links connecting MHs to their local MSSs is limited. This paper first presents a new system model for the mobile computing environment and then describes a general principle for structuring distributed algorithms in this model.

The *two tier* principle defined our approach to designing efficient distributed algorithms in this model, viz. localize the communication and data structures necessary for an algorithm within the static portion of the network to the extent possible; the core objective of the algorithm is achieved through a distributed execution amongst the fixed hosts while performing only those operations at the mobile hosts that are necessary for the desired overall functionality. Power consumption at the mobile hosts is thus kept to a minimum, and since updates to the data structures are performed at the fixed hosts, the overall search cost is reduced.

A fundamental algorithm in distributed systems viz. a token circulating in a logical ring, served as an illustrative example. Algorithm R-MH established the logical ring amongst MHs. This was shown to be inefficient in terms of search cost, power consumption at the MHs, usage of wireless links and handling disconnection of MHs. The *two tier* principle was applied to remedy these drawbacks by establishing the logical ring amongst the MSSs.

The *two tier* principle by itself is not sufficient to handle the effects of varying location of MHs. This required location management of migrant MHs and we presented three strategies: (1) search (2) inform (3) search within a local area with location updates after wide-area moves. The relative merits of the three strategies were quantitatively compared. It was then shown that mobility of a host could determine how often it was able to access the token per traversal of the ring. Since algorithm R-MH allowed each MH to access the token atmost once per traversal, we needed an equivalent functionality when the logical ring was shifted to the static segment. A scheme was presented to this end which was equally applicable to all three location management strategies.

Finally, we considered an alternative approach to handling the effects of varying location of MHs namely, replicating the queue of pending requests at all MSSs. This eliminated the need for location management strategies for migrant MHs, but increased the number of messages exchanged within the fixed network to globally order pending requests among all MSSs.

Bibliography

[1] Arup Acharya and B. R. Badrinath. A framework for delivering multicast messages in networks with mobile hosts. Technical report. Submitted for publication.

[2] Arup Acharya and B. R. Badrinath. Delivering multicast messages in networks with mobile hosts. In *Proc. of the 13th Intl. Conf. on Distributed Computing Systems*, May 1993.

[3] D. Agrawal and A. El Abbadi. An efficient and fault-tolerant solution for distributed mutual exclusion. *ACM Transactions on Computer Systems*, 9(1), Feb 1992.

[4] Raphael Alonso and Hank Korth. Database system issues in nomadic computing. In *Proc. of the ACM SIGMOD Intl. Conference on Management of Data*, June 1993.

[5] Andrew Athas and Dan Duchamp. Agent-mediated message passing for constrained environments. In *USENIX Symposium on Mobile and Location-Independent Computing*, Aug. 1993.

[6] B. R. Badrinath, Arup Acharya, and T. Imielinski. Structuring distributed algorithms for mobile hosts. In *Proc. of the 14th Intl. Conf. on Distributed Computing Systems*, May 1994.

[7] B. R. Badrinath, Arup Acharya, and Tomasz Imielinski. Impact of mobility on distributed computations. *ACM Operating Systems Review*, 27(2), April '93.

[8] B. R. Badrinath and T. Imielinski. Replication and mobility. In *Proc. of the 2nd workshop on the management of replicated data*, pages 9–12, 1992.

[9] P. Bhagwat and Charles E. Perkins. A mobile networking system based on internet protocol (ip). In *USENIX Symposium on Mobile and Location-Independent Computing*, Aug. 1993.

[10] K. Birman and T. Joseph. Reliable communications in presence of failures. *ACM Trans. Comput. Systems*, 5(1):47–76, 1987.

[11] R. Caceres and L. Iftode. The effects of mobility on reliable transport protocols. In *Proc. of the 14th Intl. Conf. on Distributed Computing Systems*, May 1994.

[12] E. W. Dijkstra et. al. Derivation of a termination detection algorithm for distributed computation. In *Information Processing Letters*, June 1983.

[13] Micheal Bender et. al. Unix for nomads: Making unix support mobile computing. In *USENIX Symposium on Mobile and Location-Independent Computing*, Aug. 1993.

[14] N. Davies et. al. Mobile open systems technology for the utilities industries. In *IEE Colloquium on CSCW Issues for Mobile and Remote Workers*, March '93.

[15] Y. Amir et. al. Fast message ordering and membership using a logical token-passing ring. In *Proc. of the 13th Intl. Conf. on Distributed Computing Systems*, May 1993.

[16] Wayne C. Grant. Wireless coyote: A computer-supported field trip. *Communications of the ACM*, May 1993.

[17] J. S. Heidemann, T. W. Page, R. G. Guy, and G. J. Popek. Primarily disconnected operation: Experiences with ficus. In *Proc. of the 2nd workshop on the management of replicated data*, pages 9–12, 1992.

[18] L. B. Huston and P. Honeyman. Disconnected operation for afs. In *USENIX sympoisum on Mobile and Location-Independent Computing*, Aug. 1993.

[19] T. Imielinski and B. R. Badrinath. Wireless mobile computing : Challenges in data management. *To appear in the Communications of the ACM.*

[20] T. Imielinski and B. R. Badrinath. Querying in highly mobile distributed environments. In *18th Intl. Conference on Very Large Databases*, pages 41–52, 1992.

[21] T. Imielinski, S. Viswanathan, and B. R. Badrinath. Power efficient filtering of data on the air. In *EDBT '94*, 1994.

[22] J. Ioannidis, D. Duchamp, and G. Q. Maguire. Ip-based protocols for mobile internetworking. In *Proc. of ACM SIGCOMM Symposium on Communication, Architectures and Protocols*, pages 235–245, September 1991.

[23] James Kistler and M. Satyanarayanan. Disconnected operation in the coda file system. *ACM Trans. on Computer Systems*, 10(1), Feb. 1992.

[24] G. Le Lann. Distributed systems, towards a formal approach. *IFIP Congress*, *Toronto*, pages 155–160, 1977.

[25] M. Maekawa. A $\sqrt{N}$ algorithm for mutual exclusion in decentralized systems. *ACM Transactions on Computer Systems*, 3(2), May 1985.

[26] Brian Marsh, Fred Douglis, and Ramon Caceres. Systems issues in mobile computing. Technical Report MITL-TR-50–93, MITL, 1993.

[27] Carl D. Tait and Dan Duchamp. Service interface and replica management algorithm for mobile file system clients. In *Proc. First Intl. Conf. on Parallel and Distributed Information Systems*, 1991.

[28] Fumio Teraoka, Yasuhiko Yokote, and Mario Tokoro. A network architecture providing host migration transparency. *Proc. of ACM SIGCOMM'91*, September, 1991.

[29] David Vaskevitch. Database in crisis and transition: A technical agenda for the year 2001. In *Proc. of the 1994 ACM SIGMOD Intl. Conf. on Management of Data*.

[30] Hiromi Wada, Takashi Yozawa, Tatsuya Ohnishi, and Yasunori Tanaka. Mobile computing environment based on internet packet forwarding. In *1992 Winter Usenix*, Jan. 1993.

[31] T. Watson and B. N. Bershad. Local area mobile computing on stock hardware and mostly stock software. In *USENIX Symposium on Mobile and Location-Independent Computing*, Aug. 1993.